WASHINGTON UNIVERSITY

CENTER FOR DEVELOPMENT TECHNOLOGY

MEMORANDUM NO. 74/2 DECEMBER, 1974

(NASA-CR-141280) A SOFTWARE IMPLEMENTATION OF A SATELLITE INTERFACE MESSAGE PROCESSOR (Washington Univ.) 60 p CSCL 09B

N75-15330

Unclas

G3/61 06533

A SOFTWARE IMPLEMENTATION OF A SATELLITE INTERFACE MESSAGE PROCESSOR

MARGARET A. EASTWOOD
LESTER F. EASTWOOD, JR.

WASHINGTON UNIVERSITY / ST. LOUIS / MISSOURI 63130

Center for Development Technology
Communications Group

Washington University
Saint Louis, Missouri 63130

Memorandum No. 74/2 December, 1974

A SOFTWARE IMPLEMENTATION OF A SATELLITE INTERFACE MESSAGE PROCESSOR

MARGARET A. EASTWOOD

LESTER F. EASTWOOD, JR.

This study was supported in part by the National Science Foundation under Grant No. EC-38871 and by the National Aeronautics and Space Administration under Grant No. NGR-26-008-054. The views expressed in this memorandum are those of the authors and do not necessarily represent those of the Center for Development Technology, Washington University, or the sponsoring agency.

TABLE OF CONTENTS

ABSTRACT

This memorandum designs network control software for a computer network in which some nodes are linked by a communications satellite channel. We assume that the network has an ARPANET-like configuration; that is, that specialized processors at each node are responsible for message switching and network control. Therefore, the task we undertake is to design the software for these processors. The purpose of the control software is to oversee network operations -- for example, to keep track of message queues and switching functions, to tag messages with addresses and insure that they arrive error-free, and to avoid disruption of network functions where a node or communications line is inoperative.

The objective of this work is to produce a software design which both efficiently uses the network communication resources and is attractive to users. To achieve the goal of efficiency, we implement the packet reservation scheme for computer communication by satellite, the most efficient of a number of methods for dynamically allocating time on a satellite channel. To gain the objective of attractiveness, we design control software which makes network operations essentially invisible to the user.

1. Introduction

Satellite communications among computer installations may find a wealth of applications. It may be justified wherever computer installations, spread widely in location, need to communicate. This need may exist if the computer installations are operated by a multi-national firm, or the computers must access a data base so large that it would be impractical to duplicate at each installation. Computer communications by satellite or other means also allows a general-purpose, or a small, computer installation occasionally to tap the superior power of a specialized, or a large, computer.

To realize the potential benefits of satellite computer networking, a number of design problems must be solved. The satellite communications hardware design and the physical configuration of the network control mechanisms and communications lines are examples, but careful design cannot cease when these hardware configurations are specified. Control software must also be designed, to oversee network switching functions, to keep track of message queues, to tag messages with addresses, to insure that messages arrive at their proper destinations in correct order and without errors, and to keep a running tabulation of inoperative network nodes or communication lines.

The software determines, to a great extent, both the cost effectiveness of the computer communications network and its attractiveness to users. To make the network run at maximum efficiency - minimum cost - the control software must make maximum use of its communications lines. It should avoid inordinate delays in operation because of failure of a network node or of a communication line. Moreover, the control operation should be fast, without an unnecessary overhead of control signals among nodes. This last point is especially important in satellite networks, in which a node-to-node communication may spend a quarter of a second in transit - a very long time in computer terms.

Attractiveness to users, as stated, is a second design criterion for network control software. To be more specific, the network control mechanism which is ideal from the user's point of view should make the networks "transparent" to the computers it serves. That is, programmers should have to do no more work to use the network than they do in employing any input-output device.

It is the purpose of this memorandum to propose a satellite computer network control software design to satisfy these constraints. To insure that the software is designed for a realistic network configuration, the memorandum briefly reviews existing computer networks, and it investigates a series of proposals for satellite computer communications schemes. In addition, it discusses example user characteristics which determine the choice of one of these schemes, because user characteristics set the criteria which the network must satisfy. We choose, after these preliminary considerations, to implement one such scheme, originally proposed by Roberts [11]. The body of the paper then presents a detailed proposal for a control software implementation of Robert's satellite computer communication scheme.

2. Current Computer Communication Networks

2.1 Introduction and History

In the past few years, operating computer communication networks have proliferated. Computer networks serve many purposes: 1) timesharing systems and remote job entry; Tymnet and Infonet (1) are two examples, 2) inquiry systems; the most common examples are the airline reservation systems, 3) database sharing and centralized processing of data; this function is often done by large corporations, and 4) resource sharing; many nets which connect university computer facilities have this purpose.

Computer networks are projected to grow at an accelerating rate. From 1972 to 1973, U.S. data communication expenditures increased 27%, to a total of $1.45 billion. The projected expenditure by 1980 is $70 billion (1). This fantastic market potential has encouraged continued research in all the areas involved: software, hardware, and communication techniques.

The first computer networks relied upon dial-up or leased telephone lines for communication. Voice lines' slow data rate, coupled with a high error rate, led to the improvement of common carrier facilities. Higher bandwidth lines and microwave links are now available from AT&T and several competitors. More sophisticated hardware, such as modems and data concentrators, improve the reliability and capacity of computer networks.

Sophistication in communication techniques specifically for computers has kept pace with the improvement in hardware. For example, a significant new communication technique for computer networks is packet-switching (2). Replacing the telephone system's traditional line switching, packet-switching is a network control technique suited to computers' short duration, high data rate communications, rather than to the slow, long-lasting connections made on the voice circuits.

2.2 The ARPA Network

Packet-switching was originally implemented in the ARPA network by the Advanced Research Projects Agency of the Department of Defense. The ARPANET currently (1974) connects 36 diverse computer centers, geographically spread from MIT to UCLA. The net includes nineteen different models of computers, so that the network has had to be designed to handle disparate operating systems, word sizes, operating speeds, interface hardware, etc.

The solution to the problem of users with nonuniform characteristics has been to connect each host computer to a minicomputer, called the Interface Message Processor (IMP). Host-dependent peculiarities only affect the individual host-IMP interface. All network message routing and protocol is handled uniformly between IMP's, independent of their host computers.

The IMP's are interconnected as a distributed network; i.e., every IMP is not directly connected to every other IMP, nor are they all connected to a central site as in a star network. Routing tables are maintained within each IMP, which specify possible paths from IMP to IMP to reach the desired destination. These tables are periodically updated to reflect congestion at an IMP, out-of-service IMP's or hosts, etc.

After a host passes a message to its IMP, the IMP breaks the message up into fixed-length packets, and sends the packets out into the net, one by one. Each IMP along the route stores the packet until it receives an acknowledgement from the next IMP in the path. The destination IMP reassembles these packets as they arrive, and then passes the complete message to its host (3).

2.3 The Aloha System

Another innovative method for computer communication is the Aloha System, developed by the University of Hawaii. Although the Aloha System is not a large-scale project like the ARPANET, it has been the basis for much of the design thinking done for satellite computer communications.

The Aloha System uses message packets like the ARPANET. Since there are few telephone lines between the Hawaiian Islands, however, the packets are sent by a radio transmitter from timesharing user terminals to the main University computer's IMP.

The user terminals are not synchronized, nor are they coordinated by extensive "handshaking" as in the ARPANET. Each terminal cán transmit at will. If several do transmit at the same time, the packets will overlap at the receiver, and are said to be blocked. Blocking is detected with a cyclic check sum scheme, and blocked packets are not acknowledged. After a random time delay, each user terminal will automatically retransmit any unacknowledged packets. (4)

2.4 Satellite Computer Communications

The logical extension of radio transmission of data was to consider how communication satellites might be used for computer communications. The cost of satellite channels has been decreasing steadily, and their use for television and telephone service has been well established. If the estimate of 2.5 million terminals in the United States by 1980 is correct, the load on the current telephone system will be severe (5). Redundant lines erected across the country by private enterprises are probably a costly and inefficient solution. Satellites might lessen the load; satellite communication costs, unlike ground lines, are independent of distance. Recognizing this fact, Western Union, American Satellite Corp., RCA, Western Tele-Communications Inc., MCI Communications Corp., and Lockeed Aircraft Corp. all applied to the FCC to operate communication satellite systems (6). Western Union, with an operating satellite, and RCA, with a launch planned in 1976, are well on the way.

The Aloha System described earlier is currently connected to the University of Alaska via NASA's ATS-1 satellite, and to Ames Research Center in California via COMSAT's Intelsat IV on an experimental basis (7).

The ARPANET is also proposing to use satellite links to decrease cost while increasing speed and capacity. Their 50 kilobit/sec leased lines cost $1,200,000 per year (January, 1973), with a capacity of 300 million packets per month. They estimate that the satellite links would cost between $1 and $3 million per year, with a capacity of 1500 million packets per month (8).

3. Proposals for Computer Communications Via Satellite

Someone once said that a satellite should not be thought of as just "a big cable in the sky". (9) Although data could be sent over the existing satellite telephone channels, much as if these lines were in fact cables stretching from point to point, inherent properties of the satellite communications channel might make a new approach desirable. Satellite channels can be designed for a much greater capacity and higher transmission rate than existing common carrier lines. Secondly, the cost of satellite transmission is independent of the distance between the sender and receiver. Thirdly, a satellite broadcasts simultaneously to all groundstations; transmission is not point-to-point. This property allows the transmission of common data to several computers simultaneously (10).

However, there is a major problem in using satellite channels for computer communications. Since communications satellite maintain a geosynchronous orbit 36,000 km. from the Earth's center, all messages arrive at their destination approximately 1/4 second after they are sent (8). This propagation delay up to the satellite and back down again is not important for voice traffic. However, most modems and line protocols are not designed to expect a delay of that magnitude. The delay causes other problems, too, such as increasing the amount of buffer space required for messages, in case one of them has to be retransmitted.

Assuming that the benefits outweigh the disadvantages, the next step is to determine a communication scheme which best utilizes the satellite channel. Although SPADE and MAT-1 are sophisticated access systems for telephone connections via satellite, they are not suited to data traffic, which does not have the same statistics as voice traffic (11). Therefore, new methods, tailored to computer requirements, have been sought.

The first schemes suggested--and rejected--were Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA). For a given channel bandwidth, TDMA allows a high burst data rate, while FDMA gives a lower, but continuous rate (12). However, both schemes have the disadvantage that time slots (TDMA) or carrier frequencies (FDMA) must be preassigned to specific ground stations. Such a fixed allocation scheme can result in wasted channel capacity; station A may have a large message queue waiting to be transmitted, while station B's slots or preassigned channels are empty.

The Aloha System's technique for multiple access--any station can transmit at any time--is a dynamic, rather than fixed, allocation scheme. It does give all stations with traffic an equal chance at the channel. Unfortunately, blocking becomes a major problem if there are many users (13).

Some improvement is obtained with the "slotted Aloha" system. Time slots are defined, but any station can attempt to use any slot. Even with this method, only 37% of the channel capacity can be used before collisions become excessive (13), and it has the disadvantage relative to "pure" Aloha that synchronization of all transmitting stations is required.

Another modification suggested by Bolt, Beranek, and Newman, is called the BBN Aloha system. In their scheme, once a station has successfully claimed a time slot, it can use that same slot, uncontested, in every cycle thereafter. When that station is done transmitting, the slot is again "up for grabs". The major problem with this method is that one or two heavy users could claim almost all the slots, and other stations would be forced to wait an unacceptable amount of time before gaining access to the channel (14).

In June, 1973, Dr. Lawrence Roberts described another plan in a paper presented at the National Computer Conference. Called "Dynamic Allocation of Satellite Capacity Through Packet Reservation", it seems to offer a more satisfactory method of allocation for computer traffic than any of the other techniques discussed (11). With this method, a ground station can reserve contiguous time slots in which to transmit a message. This is accomplished by defining a time slot cycle, which consists of several long time slots, followed by a long slot subdivided into short slots. The station first transmits a reservation during a short slot. The reservation gives the station's ID and the number of long slots to be reserved. All other ground stations record and honor the reservation. The station then transmits the message itself, using the long time slots set aside for it.

There are several advantages to this system. First, the short reservation slots are accessed as in the Aloha system, so every station desiring to transmit has an equal chance at the channel. Secondly, although collisions will occur while making reservations, the entire message will be transmitted without interruption. Thus the channel is unblocked most of the time, so a much larger percentage of the channel capacity can be used. However, a disadvantage which may be important is the increased delay before sending the message. A schematic of the procedure is shown in Figure 1.

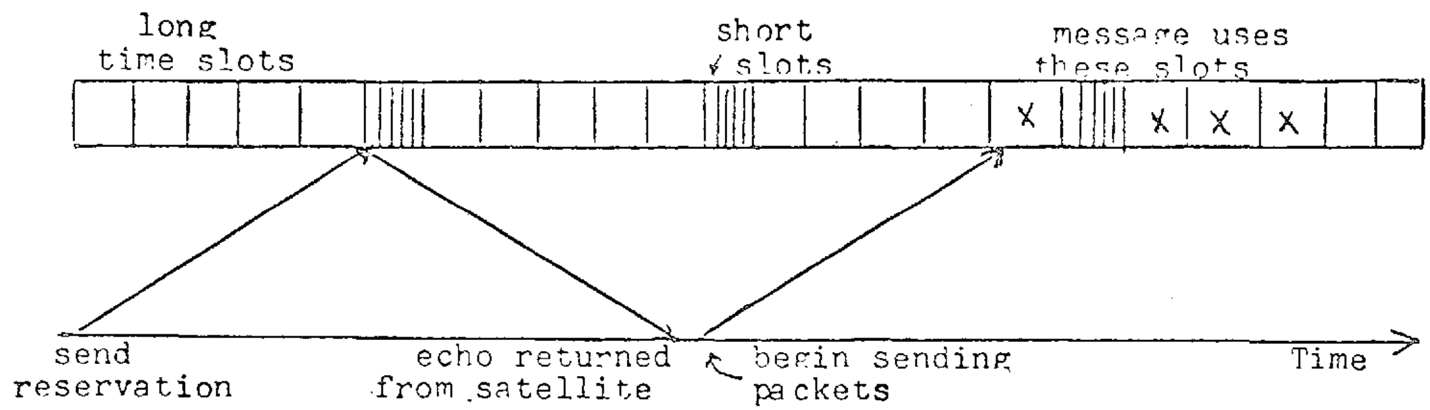


Figure 1. Schematic of the Packet Reservation Access Method

4. Influence of User Characteristics on the Implementation of the Packet Reservation Access Method

Before a software implementation of the packet reservation scheme can be designed, some consideration must be given to identifying the potential users of the system, and examining their requirements.

As stated earlier, because satellite communication cost is independent of distances, users most likely to find it cost effective are those with widely scattered installations. There are many applications which have a long distance computer communications requirement. For instance, large world-wide corporations which gather data at their headquarters, banks which transfer funds throughout the world, and offices of the travel industry, which must process airline, tour, and hotel reservations, are all potential customers.

Government agencies also transmit a vast amount of data great distances. For example, hundreds of millions of bits per day of raw data from weather satellites, the Earth Resources Technology satellite, and from manned space flights are now transmitted to ground stations in Canada, Brazil, Australia, Spain, Alaska, Maryland, and Virginia. The data is later collected and processed in Texas, California, or Maryland (15).

Universities are considering resource sharing on an international basis. A project is currently being studied to connect university computers in Japan, Australia, Hawaii, and the mainland U.S. with those in eastern Asia and the South Pacific (7).

These applications suggest several constraints on the software implementation of channel access. First of all, the software should be as independent as possible of any specific computer hardware. This independence can be achieved to some degree by using IMP's (or SIMP's, for satellite IMP).

A second factor in the software design is whether each network will lease an entire channel, or if each channel will be shared by a variety of users as their needs dictate. A network with its own satellite channel could pick its own packet format and channel protocol. The shared channel situation is more likely, however, since few applications could keep the channel busy with continuous computer transmissions. A shared channel being used as a common carrier by many networks must have a much more general protocol and format.

A third complication which should be considered in the software design is whether or not all the nodes of a given network are ground stations. They may not all be, for instance, if two existing ground networks are connected via a satellite link. In that case, one localized ground network might route all its long distance messages to a single ground station. After traversing the satellite link, the messages would be distributed from the receiving ground station to the other land network. This implies that the packet format, etc., must work equally well for either ground or satellite links of a network.

Two other objectives of the implementation--obvious, but important--are system reliability, and high throughput for all users.

With these objectives in mind, the details of the implementation can be tackled.

5. Implementation of Roberts' Packet Reservation Scheme

To implement Roberts' packet reservation system, this paper proposes an ARPA-like philosophy in which SIMPs (satellite IMPs) are used as the message handlers. In an operation which is transparent to the host computer, a SIMP accepts messages from its hosts, breaks them into packets, reserves slots for them on the channel, transmits the packets, and ascertains that they were received correctly. At the other end, a SIMP receives the packets, combines them back into a complete message, and passes the message on to its host. Although the host-SIMP interfaces may vary, the satellite accessing portion of all the SIMPs are identical.

Those aspects of Roberts' original paper which were developed specifically to be used in conjunction with the existing ARPANET have been ignored or changed, in some cases. The implementation presented here is for use of the satellite channel as a common carrier, available to any network.

5.1 Data Formats

The first item to be specified in this plan is the message packet format. Figure 2 shows a possible format. The packet consists of a header portion, and a text portion, with cyclic checksums for each. The header must contain an identification of the source and destination of the packet. To decrease the probability of a lost or duplicate packet, it is also desirable to include a message and packet number in the header. By limiting the number of ground stations using a given channel to 255, and the maximum number of packets per message to 16, the packet header size can be 32 bits:

8 bits	destination
8 bits	source
4 bits	packet type code
8 bits	message number
4 bits	packet number

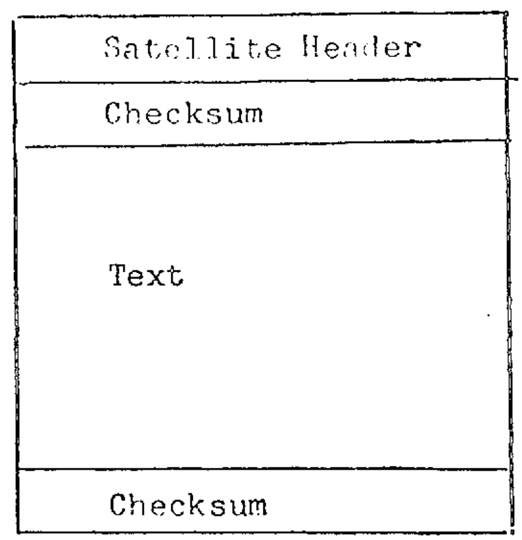


Figure 2. Possible Message Packet Format

The packet type code will be described later. Message and packet numbers are assigned sequentially by the SIMP sending the message.

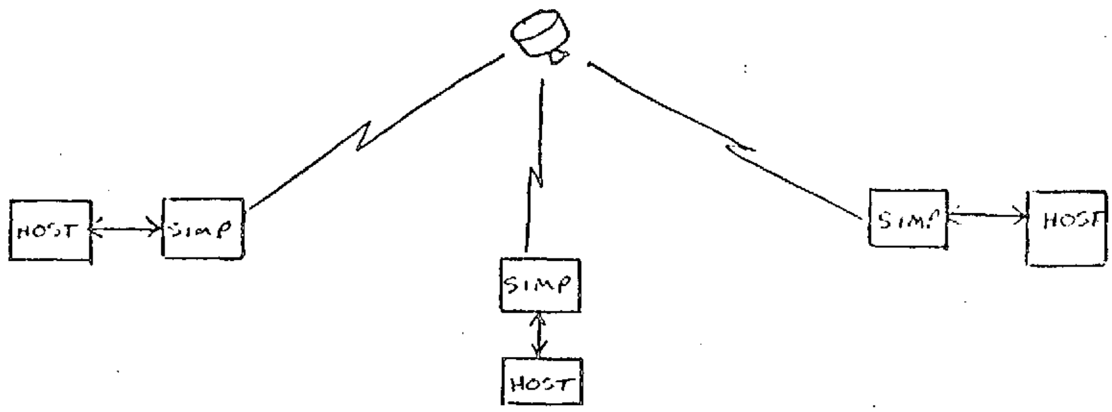
The source and destination numbers normally specify the host computer from which the packet was sent, or for which it is intended. When the host is in a ground net, however, a "gateway" computer into the net should be specified instead. This removes the burdens of local (as opposed to satellite) protocol from the SIMP. Three possible network configurations are shown in Figure 3. Notice also that, because the SIMP removes the packet's satellite header, network headers must be placed in the text portion of the packet, as shown in Figure 4. By giving a destination number of 255 (all ones), a packet can be directed to all SIMPs simultaneously.

A universal set of control characters for use between SIMPs must be agreed upon for such things as "start of satellite header", "end of transmission", and the like. Currently, there is no standard; EOT in EBCDIC is 00011011, whereas in ASCII it is 00000100 (16).

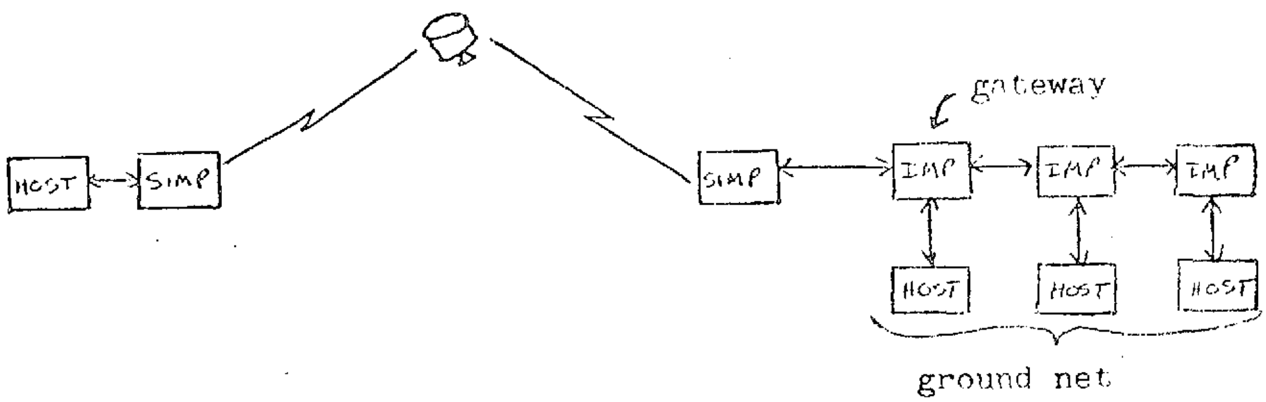
The text portion of the packet can contain anything. Its contents are not examined by the SIMPs.

Since the packet reservation method uses both large and small channel slots, a small slot packet format must also be designed. Roberts uses small packets of 24 bits for reservations and acknowledgements. The implementation presented here has found it useful to use 32 bits, and to define four new types of small packets: request for status, response for status, request for queue length, and response for queue length.

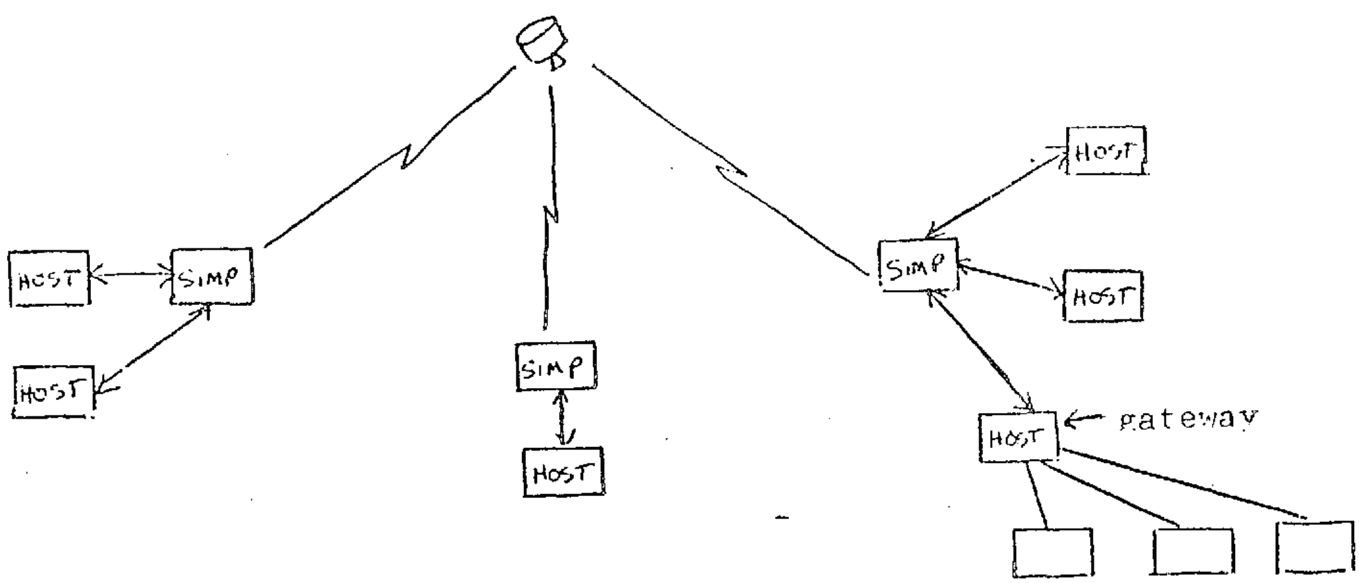
The request for status packet will be sent by one SIMP to another, if it has been waiting an excessive amount of time for either an acknowledgement or more packets within a given message from that SIMP. If the second SIMP is still operating, it should reply with a response to status packet. Obviously, if it is not operating, it will not send a response.



A. All Nodes Are Ground Stations



B. A Gateway into A Ground Net Has Been Specified



C. SIMP Handles Multiple Hosts And/Or Ground Nets

Figure 3. Possible Satellite Computer Network Configurations

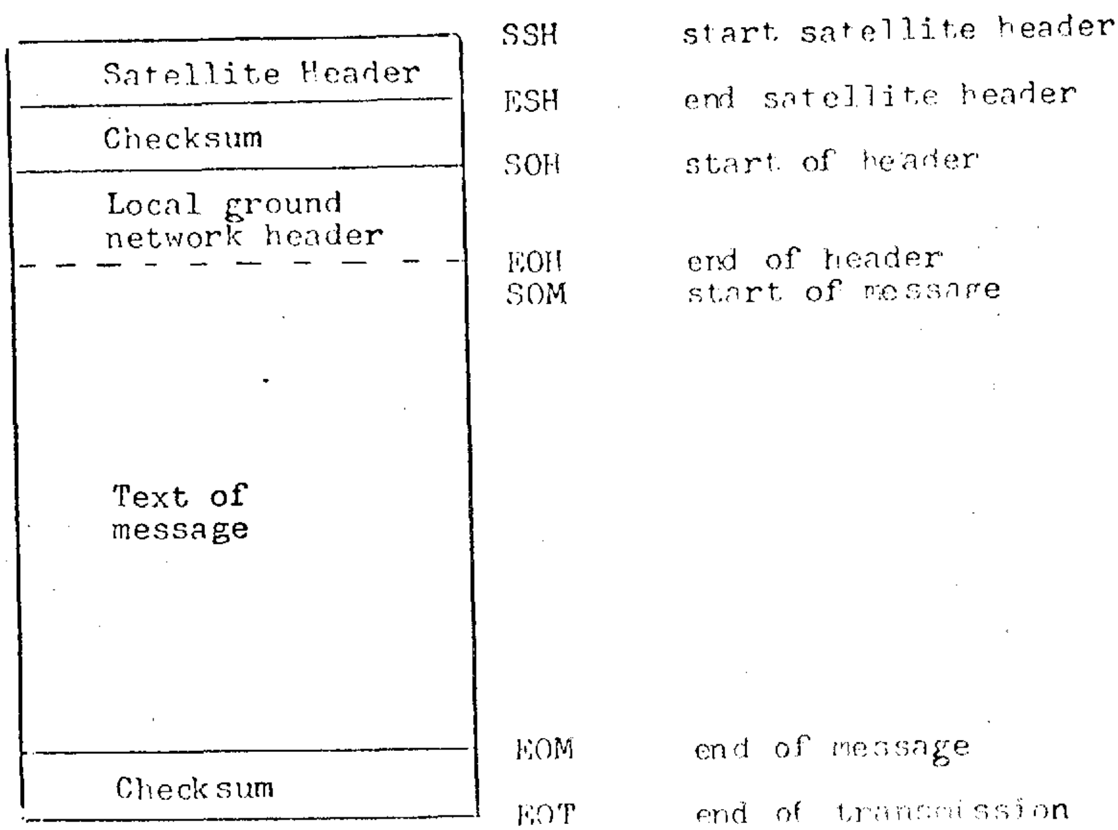


Figure 4. Message Packet Format and Control Characters

The response to status packets can also be used without a prior request. By specifying a destination number of 255, a SIMP can send a response to status packet to all the other SIMPs, to inform that a host is down (for maintenance, etc), or that one formerly down is now back up.

The request for queue length packets may be needed when a SIMP first tries to gain access to the satellite channel. If several SIMPs are already transmitting on the channel, the newcomer has no way of knowing how many slots they have already reserved. Since all operating SIMPs keep a running account of the reserved queue length, any of them could answer with a response to queue length packet. For this implementation however, we assume that a particular SIMP has been assigned to transmit the answer.

The formats of the small packets are shown in Figure 5. They have only a header, no text. The packet type codes (bits 17-20) which identify them are given in Figure 6.

Each packet, whether large or small, is preceeded by an 80 bit synchronization pattern. This is a standard way to prepare the hardware to recognize the data. It is used, for example, in IBM's bisynchronous communication (BSC). The sync pattern also serves as a buffer zone, in case the packet is not transmitted at exactly the nominal time. This is important because the ground stations are not all equidistant from the satellite, and transmission times vary.

Based on the design outlined, the total size of a large packet is 1350 bits: 80 for synchronization, 24 for each of the checksums, 32 for the header, and 1190 for text and control characters. The small packets are 248 bits. Although the header and checksum lengths add up to only 56 bits, the information in the small packets is important enough to successful operation of the system that three copies of them are sent to cube the probability of error (8). The additional 80 bits is for the

	Large Packet Header	Reservation Packet	Acknowl-edgment	Request For Status	Response To Status	Request or Response To Queue Length
8 bits	Destination	Dest.	Dest.	Dest.	Dest.	Dest.
8 bits	Source	Source	Source	Source	Source	Source
4 bits	Packet type code	Packet type code	Packet type code	Packet type code	Packet type code	Packet type code
8 bits	Message #	Number of slots reserved (uses only 4 bits)	Message #		01 = up 10 = down (uses only 2 bits)	Length (respons uses all 12 bits)
4 bits	Packet #		Packet #			

Figure 5. Small Packet Format

Code	Packet Type
0000	Large Message Packet
0001	Reservation
0010	Positive Acknowledgment
0011	Negative Acknowledgment
0100	Request For Status
0101	Response To Status
0110	Request For Queue Length
0111	Response To Queue Length

Figure 6. Packet Type Codes

sync pattern. Using these packet lengths, one large time slot can be divided into five small slots. The resulting channel access pattern is five long slots, followed by five short slots, as shown in Figure 7. Assuming a 50,000 bit/sec rate, a long slot is .027 seconds, and one pattern is .162 seconds.

The format of the data passed between a SIMP and its host must also be defined. Very few conventions are needed. Uniform control characters are not required, since the I/O routine between the SIMP and host is host-dependent, and will recognize those used by the host. Synchronization patterns are not necessary, either. Transmission can be done in start-stop mode, character by character. Using start-stop mode with the host allows the SIMP to respond more quickly to an interrupt from the satellite, without losing data. Finally, the host does not have to worry about word size. The SIMP will pack and unpack characters if necessary. The only restriction on data between a SIMP and host is the message length. The satellite header format already chosen limits the length of a message to 16 packets. Therefore, the host must create messages $\leq$ 19,000 bits long. The first 8 bits must be the destination number, so that the SIMP can create the appropriate header.

5.2 Hardware Functions

With the host-SIMP and SIMP-satellite data formats defined, the next step is to determine how the data will be manipulated. Most of the processing will be done by software, but several functions can be simplified or completely accomplished by hardware. For this implementation, we have designated hardware to do the following:

1) Insert and remove checksums
2) Transfer data from the SIMP's memory to the transmitter
3) Transfer data from the receiver into memory
4) Identify packets to be processed by this SIMP

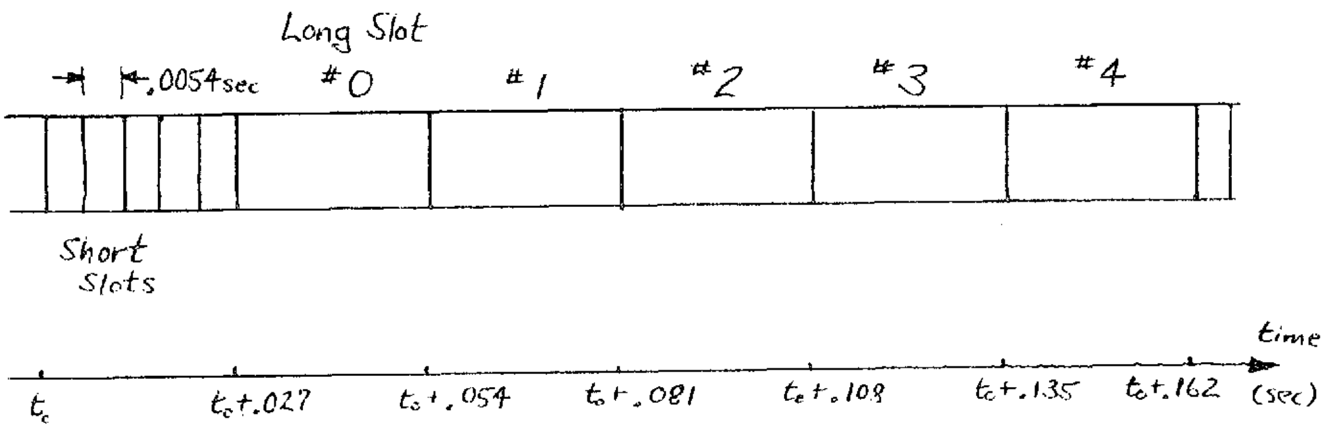


Figure 7. Channel Access Pattern

Checksum hardware, item (1), is very common. In this case it must be designed to insert a checksum after every ESH and EOT control character. On the receiving end, it has to recompute the checksum. Unlike some ground networks, the checksum hardware cannot send back the ACK or NAK itself. Instead, all acknowledgements must be formatted into small packets, and only transmitted during short time slots. Therefore, it is assumed in this implementation that the checksum hardware appends a flag to the end of the header and text portions, indicating whether or not an error was detected. The SIMP will check the flag before using the data.

Items (2) and (3) in the list above were necessary since the SIMP may be sending and receiving packets simultaneously. At the time a SIMP makes a reservation, it does not know if another SIMP will be sending it packets during its transmit time. By having direct memory access (DMA) in the SIMP, nearly automatic transfers of data to the transmitter and in from the receiver can be realized. This will be discussed further in Section 5.5.

Item (4) becomes more of a necessity if bit rates across the satellite link are high. Recall that the satellite broadcasts every packet to SIMP. At a 50K bit/sec rate, with a full channel, a packet will arrive at each SIMP every .027 seconds. Although a SIMP may be able to handle all packet processing fast enough, it is a waste of computer power for a SIMP to receive, examine, and then discard packets not intended for it. Instead, the receiver hardware could be designed to pass on to the SIMP only those packets destined for one of its hosts, as shown by the destination field in the header.

This approach works well for the long packets, but is not appropriate for the short packets. Every SIMP must listen to the reservations made by others, and to responses to status packets. In addition, a SIMP has to listen to the echo back from the satellite of every short packet it sent, to check for blocking. These packets have the SIMP's ID in the source field of the header, rather that in the destination field.

To handle this problem, it is very convenient to have the satellite itself send out a timing signal every .162 seconds, to signify the beginning of the group of short time slots. This timing signal will keep everyone synchronized, reset the receiver to non-selective mode, and will cause an interrupt in the software. The software can later put the receiver back into selective mode.

5.3 Message Transfer From a Host to its SIMP

With the data formats and special hardware defined, the design of the software itself can begin. The first function a SIMP has to perform is to accept messages from its hosts and format them into packets. The SIMP polls each host in turn, to see if any of them have data to be transmitted. If one of them does, the SIMP stops polling, and prepares to read in the data from the host.

Figure 8 shows the data structure which is used as messages from the host are being assembled into packets. The assembly header cells are always present, with one for each host. The blocks of core used to store the packets are obtained one at a time, as needed, from an available space list (ASL). Since the data is transferred character by character, a field is maintained listing the number of bits stored so far in the last packet.

Before the SIMP reads in any characters from the host, it must determine if the data will begin a new message (indicated by a null forward pointer for the host's assembly header cell), or if it is more of a partially assembled message. If it begins a new message, the SIMP must get a block of core from the ASL, assign the next sequential message number, and format the satellite header. This will be packet number zero of the message. Then data can be moved into the text portion. As a

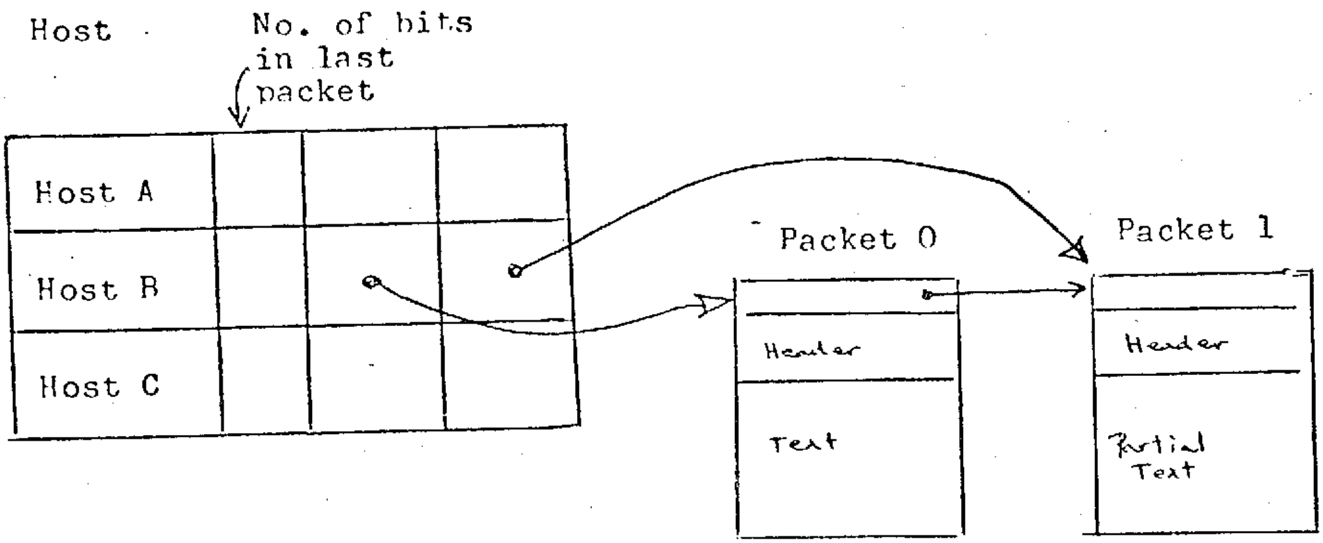


Figure 8. Data Structure Used for Assembling Messages into Packets; Assembly Header Cells

packet is filled, another block of core must be added to the list. This packet will get the same header, except for the packet number field, which increases by one.

The SIMP will continue to read in data from this host until 1) the SIMP has read in the entire message (recall that there is a maximum allowable length) or 2) an interrupt occurs (usually from the satellite). When the SIMP again resumes servicing the hosts, it does not begin where it left off, even if it was in the middle of a message. Instead, it polls again, starting at the next entry in the polling list, following the one it serviced the last time. In this way, each host should get approximately equal service. If one host requires priority service, its address can appear several times in the polling list, so its turn occurs more frequently.

When an entire message has been converted to a string of packets, the last packet header is marked by setting the leftmost bit of its message number field to a one. The packets are then ready to be transmitted over the satellite channel. Transmission is described in Section 5.5.

5.4 Packet Receipt and Acknowledgment

Ignoring for the moment how the packets are sent over the satellite channel, this section discusses what happens to the packets once they arrive at the other end. Basically, the receiving SIMP will read in the packets, strip off the satellite headers, assemble the text back into a message, and pass the message to the destination host.

As each packet arrives, the first thing the SIMP must do is examine the checksum flag to see if the packet arrived without error. In the original ARPANET, every packet is acknowledged with either an ACK or NAK. This is impossible with the satellite channel as we have designed it. For instance, if all five large slots contained packets, then all five small slots would be required for their acknowledgments. There would be no small slots available for making reservations, etc. Instead, it seems more efficient for the receiving SIMP to send only a single positive acknowledgment after the entire message, to show that it was received correctly.

A NAK, however, can be sent as soon as an error condition is detected in one of the packets. There is no reason to wait until the last packet has been received. There are several conditions which result in a NAK: 1) a checksum error, or 2) an out-of-sequence message or packet number, or 3) a long time interval since the last packet of an incomplete message arrived. This third condition is not necessarily an error, if the channel is very busy. However, it could also indicate a breakdown at the sending SIMP, or a checksum error in the satellite header such that the receiver hardware, in selective mode, did not recognize the destination field.

Due to the long time delay when returning acknowledgments, other packets of the same message may arrive after the bad one. Consider Figure 9. Assume a SIMP begins transmitting an 8-packet message at time A. The receiving SIMP sees it at time B, and detects an error in the third packet at time C. Since acknowledgments require small slots, the NAK cannot be sent until time D. By the time it is finally received, at E, the rest of the message has already been sent.

A decision has to be made as to what to do with the later packets, numbers 3 through 7. The SIMP could add them to the string of packets, with a flag set to show that the message is waiting for a packet to be retransmitted. That retransmission could take a long time, though, and the later packets will be tying up core until it arrives. Also, the processing becomes more cumbersome if several packets are missing, say 3, 4, and 7. Each time a retransmitted packet does arrive, the whole string must be checked to see if the error flag can be removed yet, or

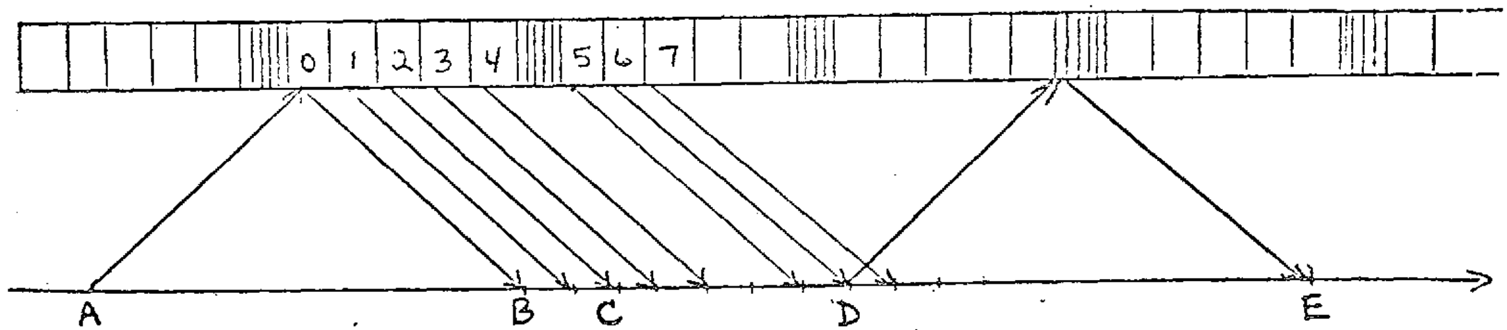


Figure 9. Time Sequence Example for Packet Received in Error

if there are still some missing. For these reasons, and because frequent large packet transmission errors are unlikely, (for a typical bit error rate of 10^{-7}, errors will occur every 10,000 packets or so), this implementation chooses instead to have the receiving SIMP save the correct packets received before the erroneous one, but throw away any packets (of the same message) which arrive after it. When the sending SIMP receives the NAK, it will retransmit all the packets with numbers greater than or equal to the packet number given in the NAK (see Figure 5 for the NAK packet format).

Figure 10 shows the data structure used to reassemble packets into messages. There is a permanent header cell for each host. Since a given host might be receiving messages from several different sources, secondary header cells, one per source, are also included. These secondary headers are not permanent; they only exist while packets are being collected. The message number and source fields are self-explanatory. The packet number field contains the last number received. The status field can have two values; a zero means everything is OK. A one means a NAK has been sent for the packet number given in the number field, and no later packets are to be added to the string. When the retransmitted copy of that packet is received, the status bit will be changed back to a zero. The time field is used to detect an excessive delay since the last packet arrived. The timer field initially has a value of zero. Periodically, a utility program adds one to it. The count will continue to accumulate until another packet is received, at which time it is again reset to zero. If the count ever reaches a maximum value, the status bit is set to one, a NAK is sent out, and a request for status packet is also transmitted to the sending SIMP.

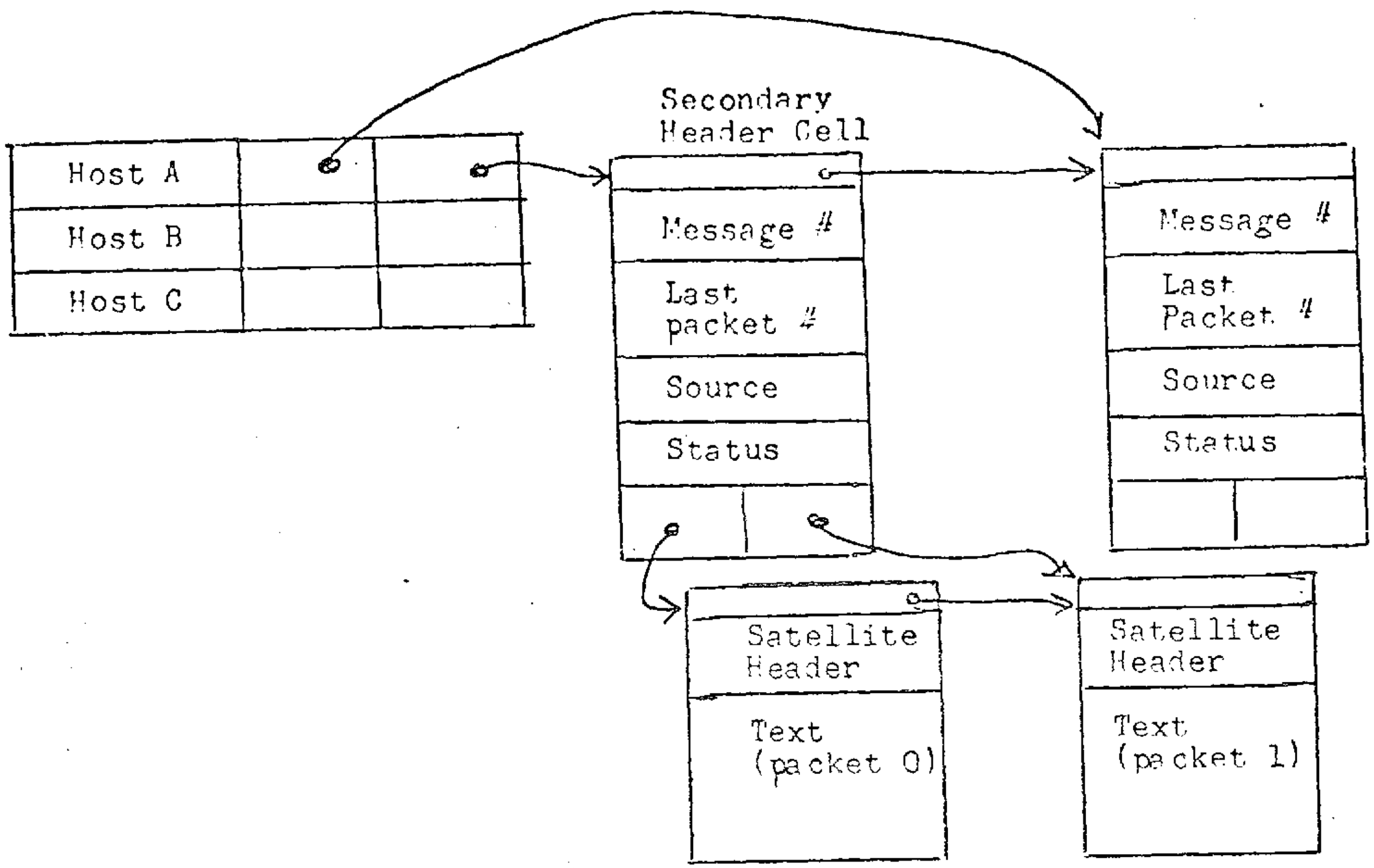


Figure 10. Collecting Packets from The Satellite

When all packets comprising a message have been received, they are moved to the "sent to host" list, shown in Figure 11. The secondary header cell is returned to the ASL. Transmission to the host is done character by character, so a field indicating the next character to be sent is maintained. The satellite header portion is not sent to the host. It could have been discarded earlier, but then two sizes of core blocks would be needed on available space lists, and the extra 32 bits is not really significant.

5.5 SIMP to SIMP Transmissions via the Satellite

The SIMP-host and host-SIMP interfaces have been described in the preceeding sections. We will now address the SIMP-to-SIMP transmission via the satellite.

Consider again the sending SIMP. It has read in a message from a host, and formatted it as a string of packets. This completed string is ready for transmission. It is desirable that the packets be sent right away. However, under some circumstances to be discussed later, there may be a delay.

When the packets can be sent right away, they are delinked from the host's assembly header cell, and linked to the end of the "ready to be transmitted" list, shown in Figure 12. It is from the front of this list that the SIMP gets packets to transmit. After the SIMP sends a packet, the packet is moved from the "ready to be transmitted" list to the "waiting for an acknowledgment" list. The acknowledgment header cells, shown in Figure 13, are not fixed. The destination fields are filled in as needed.

When the packets cannot be sent right away, it could be for either of two reasons: 1) the previous message has not been acknowledged, or 2) the destination host or SIMP is down.

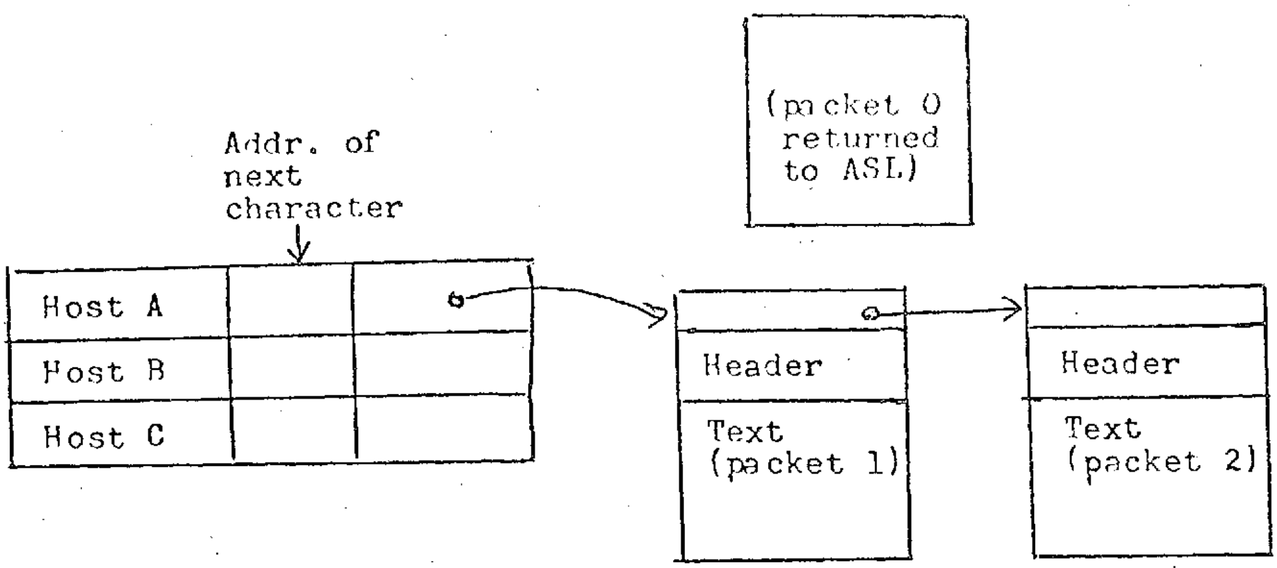


Figure 11. Messages Being Transferred from SIMP to Host

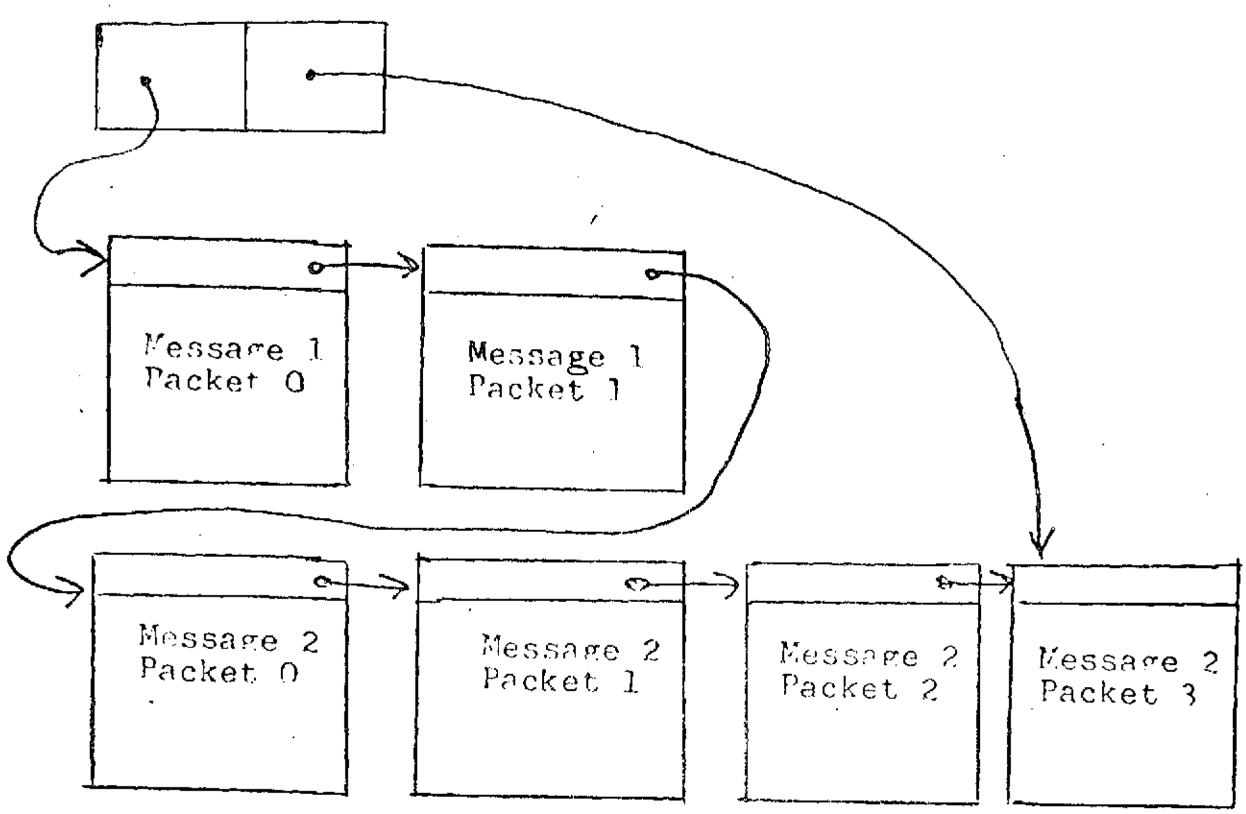


Figure 12. The "Ready To Be Transmitted" List

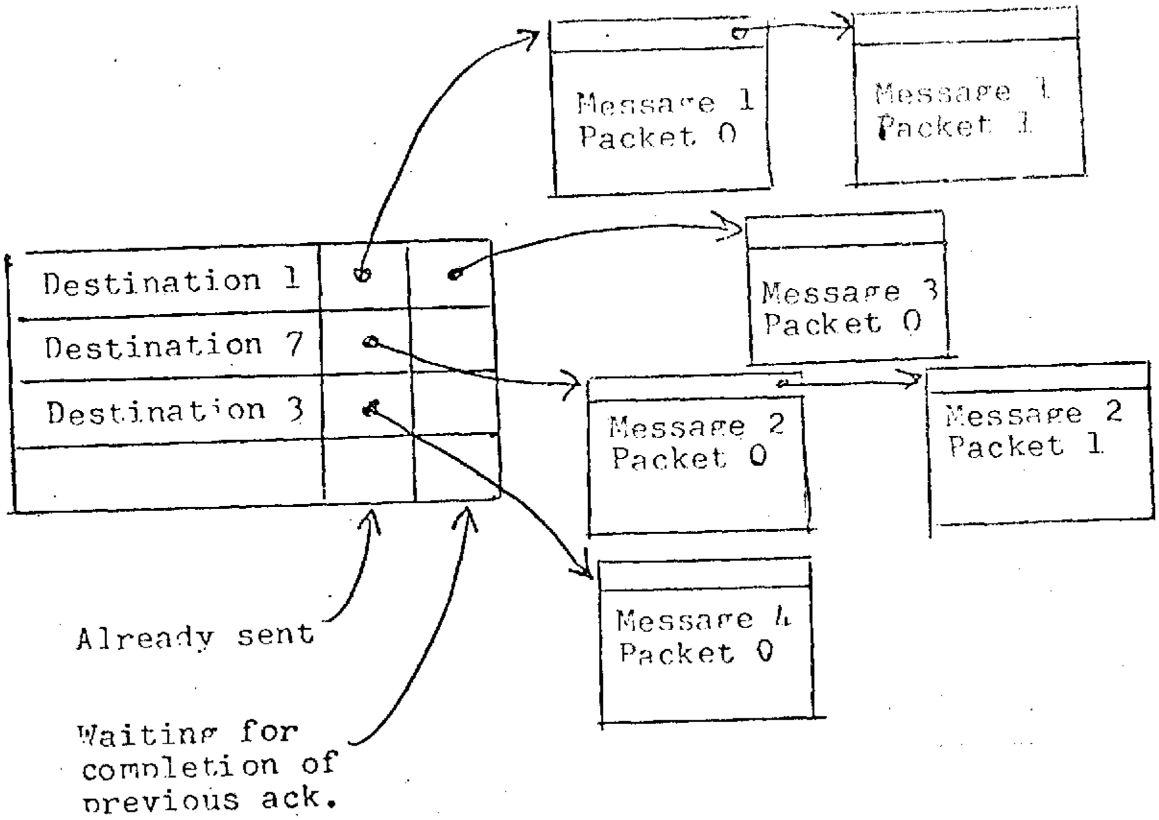


Figure 13. The "Waiting for An Acknowledgment" List

Recall that in Section 5.4 we decided to throw away any packets in a message which followed an erroneous packet. Unfortunately, this decision also implies that if message 1 had an error, and if a second message from the same source and for the same destination was sent before the NAK was received, the second message would also have to be thrown away. Otherwise, the destination host would get the messages in the wrong order. This reordering would cause a serious problem, because the messages could be portions of a sequential file or computer program. Since the packet satellite headers containing the sequence numbers are stripped off by the SIMP, the host would never know of the inversion. It is conceivable that the receiving SIMP could save message 2 until message 1 has been retransmitted, but as Figure 14 shows, the problem can soon snowball.

A more reasonable solution is not to let the sending SIMP transmit the second message until it has received a positive acknowledgment for the first one. This can be done by linking the packets of the second message to the acknowledgment header on whose completion they are waiting (shown in Figure 13), instead of attaching them to the "ready to be transmitted" list. They will stay there until the previous message has been positively acknowledged. Then the waiting string of packets will be transferred to the "ready to be transmitted" list. (The acknowledged packets are returned to the ASL).

If too many messages are waiting at the sending SIMP, the SIMP can temporarily discontinue polling its hosts for new messages until the backlog is cleared out. (Note that this wait is only required if several messages are from the same source to the same destination. There is no delay for packets to a different destination.)

The second condition under which packets would not be sent right away is if the destination host (or SIMP) is down. This condition could

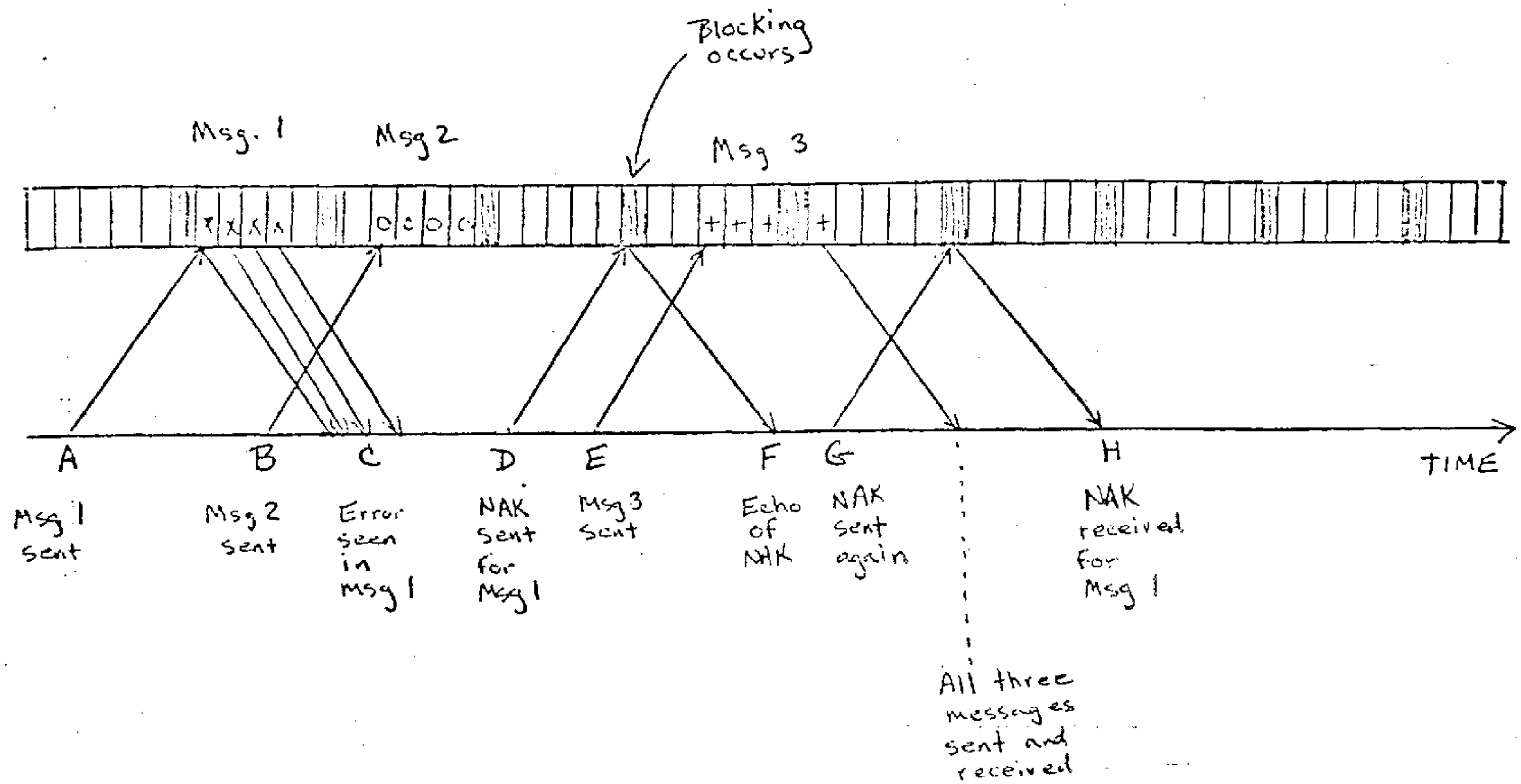


Figure 14. The Problem of Message Inversion

be recognized by keeping a Host Status Table (HST), Figure 15. This table would have two bits for every possible destination. A value of zero means the host is up and operating properly. A value of one means the host is down. A value of two means the status of the host is unknown.

Before moving a string of packets to the "ready to be transmitted" list, the SIMP would examine the HST entry for the destination host. If it is zero, the SIMP moves the packets. If it is one, the SIMP releases the packets to the ASL, and sends an indicator to the source host. If the HST entry is two, the packets are temporarily linked to the "waiting for an acknowledgment" list, as in the previous example. In addition, a request for status packet is created and sent out. If and when a response to status is received, the packet string is handled as described above. If no answer is received within a specified amount of time, the HST entry is set to one, and the packets are returned to the ASL.

Before packets on the "ready to be transmitted" list can be sent to their destination via the satellite, the SIMP must make a reservation for them. Recall that reservations are made by transmitting a reservation packet during a short time slot. Also recall that no more than eight large slots can be reserved by any given reservation packet. If two SIMPs both try to use the same small slot for their reservation packets, both reservations are blocked and must be retransmitted.

Under worst-case conditions, blocking can cause a very long time delay between the initial attempt and final success in reserving large slots, and it can be a source of confusion in reservation handling design for the SIMP. Consider the example given in Figure 16. At time A, a string of six packets (message 1) is moved to the "ready to be transmitted" list. At time B, the SIMP sends out a reservation for six large slots. At time C, four more packets (message 2) are also moved to the "ready to

Dest. 0	Dest. 1	Dest. 2			
00	00	00	10	00	00
11	00	00	00	00	00
00	00	10	00	00	00

Figure 15. Host Status Table

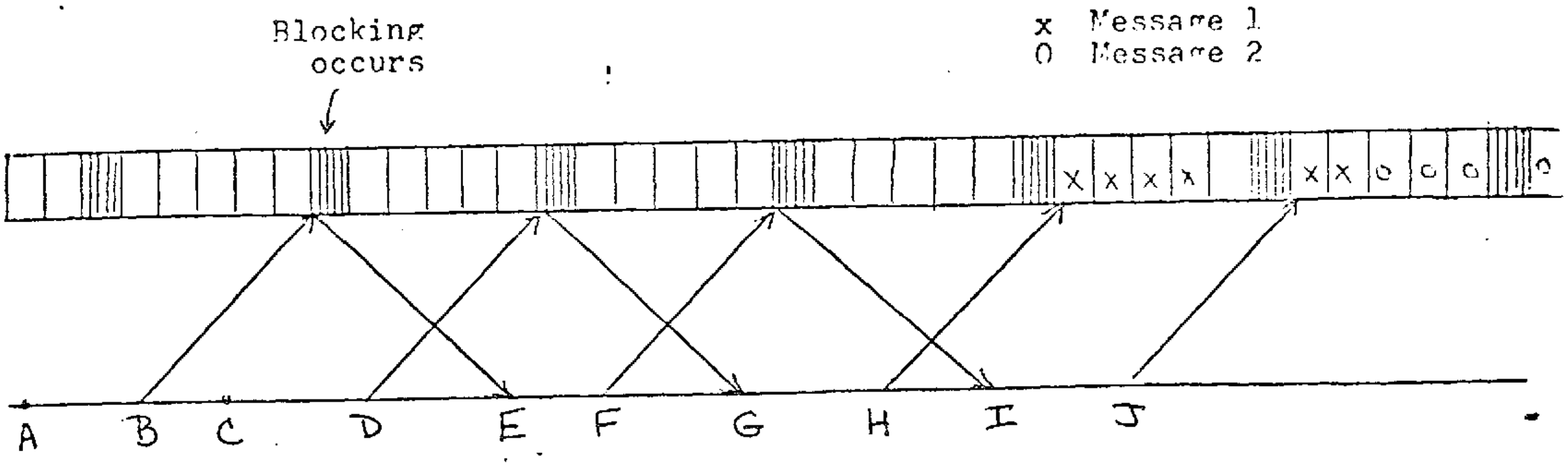


Figure 16. Example of Time Delay Caused by Blocking

be transmitted" list, so at time D the SIMP sends out another reservation, this one for four slots. At time E, the SIMP finds that the first reservation was blocked and must be retransmitted. This it does at time F. No slots are reserved as yet. At time G, however, the SIMP determines that the second reservation was transmitted without interference, so four slots are now reserved for this SIMP. Although the second reservation was originally intended for message 2, messages cannot be sent out of order. So beginning at time H, the first four packets of message 1 will be transmitted. If no more blocking has occurred, the other six slots are reserved at time I. The last two packets of message 1 and all four packets of message 2 will be transmitted starting at time J.

To keep this sequence of events unscrambled in the software implementation, two linked lists containing reservations are required. The first is the "small packets to be transmitted" list. Reservation packets on this list are waiting to be transmitted. The second list is the "pending small packet" list. Small packets on this list, including reservations, have been transmitted, but the SIMP does not yet know if they will be blocked. The order of the packets in the list is the same as the order in which they were sent. Therefore, as each echo is received from the satellite, the corresponding pending packet is either released to the ASL (in the case of reservations, the reservation was successful), or it is moved back onto the "small packets to be transmitted" list (if the small packet/reservation was blocked and must be retransmitted). Figure 17 shows the two lists.

In some cases, the small slots can be used more economically if a single reservation packet reserves large slots for several of the SIMPs messages. For example, if three messages are ready to be transmitted, three separate reservation packets (each requiring a small slot) will be

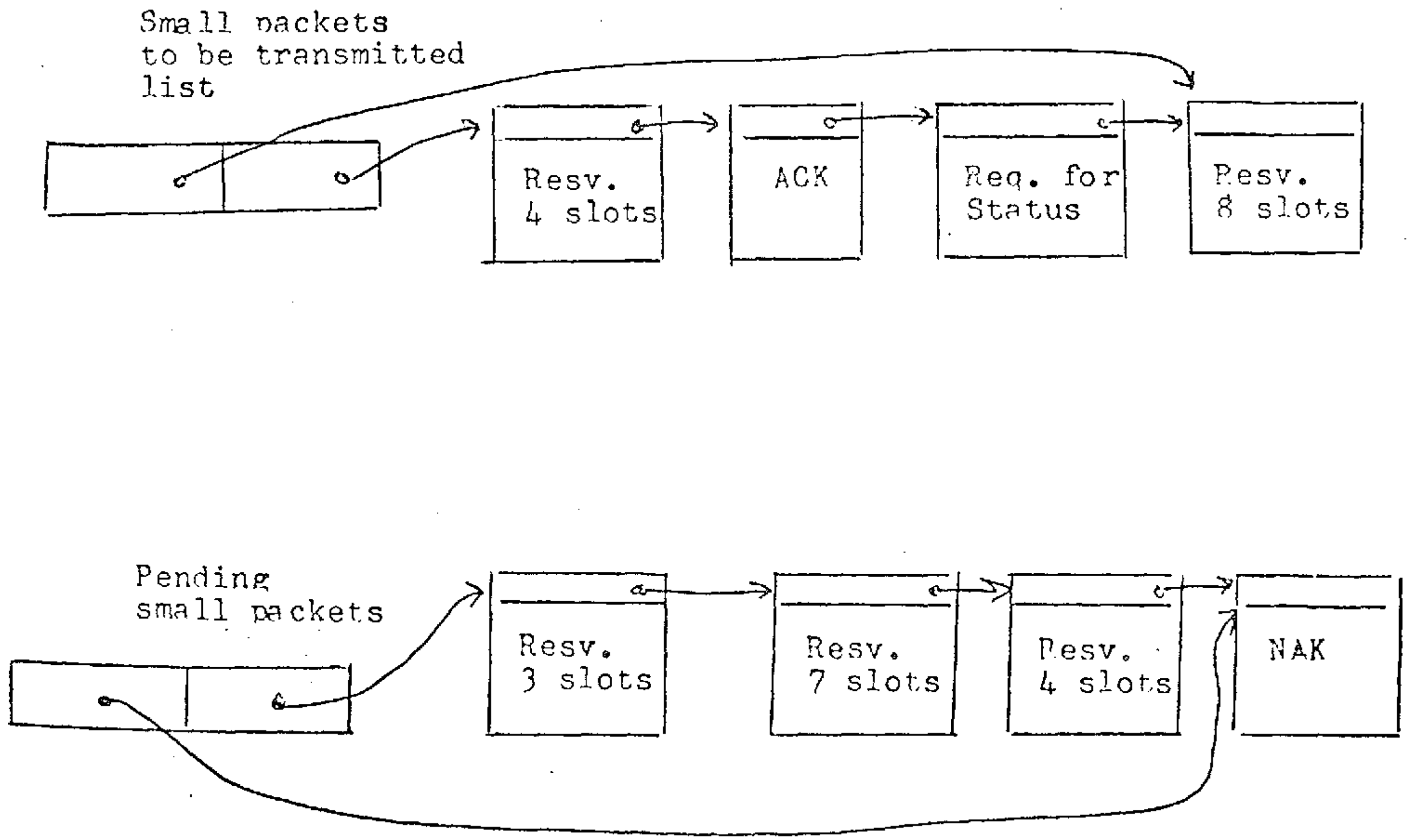


Figure 17. Small Packets Lists

sent out. However, if the messages are only two packets long, a single reservation for six packets would suffice.

This reservation consolidation can be done each time a reservation packet is about to be added to the "small packets to be transmitted" list. If the last reservation already on the list (if any) is requesting less than eight slots, the new quantity being requested can be added to it. If the sum is greater than eight, the first reservation should reserve eight slots, and the second reservation should reserve the remaining slots. The new reservation is now the last on the list, and can participate in the next consolidation.

It is not enough simply to know that slots have been reserved; the SIMP must also know <u>when</u> its slots occur. It derives this information by maintaining a queue count, which tells how many large slots are currently reserved. The queue count is modified each time new reservations are made, that is, as the echoes of the small slots arrive at the SIMP. Small slot echoes fortunately arrive immediately after the satellite's synchronization signal, mentioned in Section 5.2. Therefore, the interrupt caused by this signal will be used by the software to initiate queue updating.

Once the SIMP knows when its slots are, it must compute the correct packet send times, so that the packets will arrive at the satellite during those slots. Figure 18 illustrates the relationship of slot time to send time. Satellite interrupts will occur at times A, B, and C. If a reservation is received after time A, and the queue count is zero, the SIMP's slot will be at time E. The SIMP must send the packet at time D for it to arrive at the satellite at time E.

The time it takes a packet to travel from the SIMP to the satellite is a constant for a given SIMP. It varies between SIMPs, however, depending

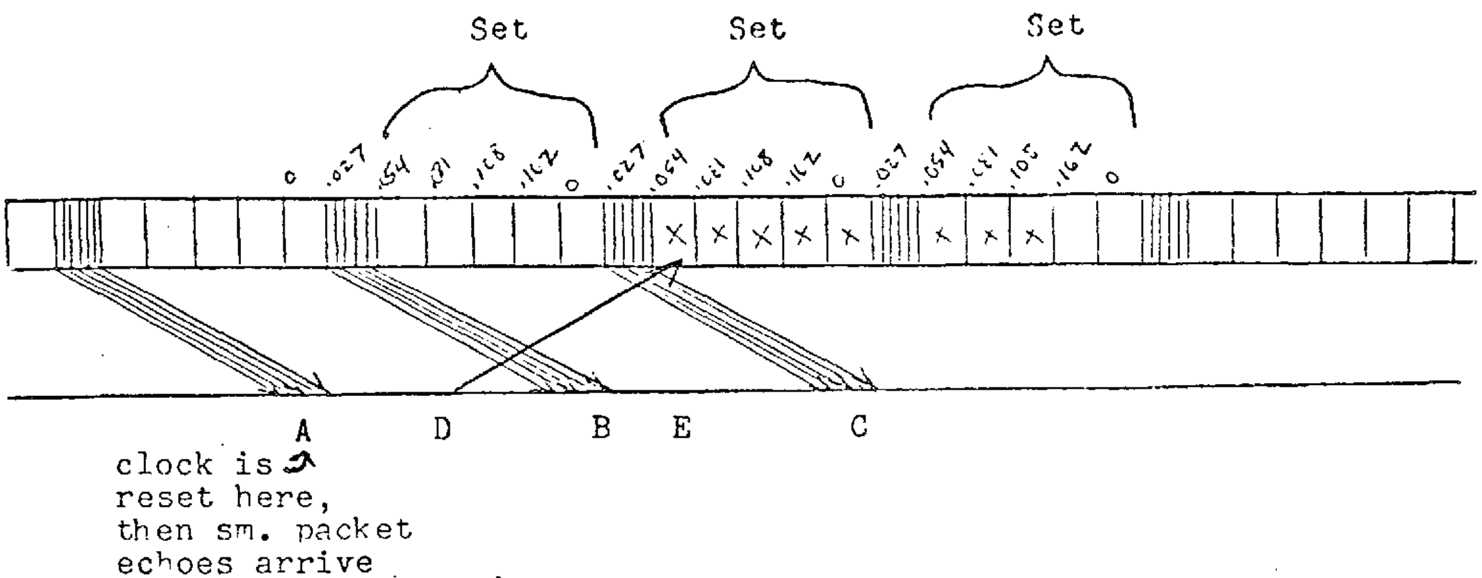


Figure 18. Relationship of Slot Time to Send Time

on the geographical location of the SIMP with respect to the satellite. The Send Time Constant Table, shown in Figure 19, contains times based on the propagation time. The Send Time Constant Table will be used to work backwards from the slot time to compute the send time.

Accuracy considerations require that a modular timekeeping procedure be used by SIMPs. Representing the send times in the software as milliseconds past midnight, or hours, minutes, and seconds, would require a very accurate 24 hour clock within the SIMP. Such a clock is not necessary if a modular form for time is used instead.

SIMPs will count time in modules called "sets". A set is a group of five contiguous large slots, as shown in Figure 18. Send times and slot times can be specified in terms of the number of complete sets, and the time within a partial set. For example, in Figure 18 the slot at time E is one set plus .54 seconds (a partial set) from A. The clock used for send time comparisons will only have to run from one synchronization signal to the next.

The required send times, in this modular form, can be stored in the Send Time Table shown in Figure 20. This table is a circular list with an IN pointer (which tells where to put the next entry), and an OUT pointer (which shows the next entry to be compared against the clock). As slot reservations are made, the SIMP calculates the send times needed to use those slots. Then one entry is made in the table for each slot. (One entry per large slot reserved eliminates the problem of knowing when to skip over the small slots. That problem would arise if only the start time and reservation quantity were stored.)

The algorithm for calculating the required send times and updating the queue count is given below. In it, the next available slot (which is also reachable at this time) is computed first, in terms of complete

Set Increment	Time Within A Set
-1	$.027 \pm \delta$
-1	$.081 \pm \delta$
-1	$.108 \pm \delta$
-1	$.162 \pm \delta$
0	$.0 \pm \delta$
0	$.027 \pm \delta$
0	$.081 \pm \delta$
0	$.108 \pm \delta$
0	$.162 \pm \delta$
1	$.0 \pm \delta$
1	$.027 \pm \delta$
1	$.081 \pm \delta$

For a single packet, only these entries are needed.

If the reservation was for multiple packets, these entries are needed.

Figure 19. Send Time Constant Table

	Complete Sets	Time Within A Set
OUT →	0	.162
	1	.081
	1	.108
	3	.108
IN →		

Figure 20. Send Time Table

sets and a slot number within a partial set. The slot number within a partial set is used as an index into the Send Time Constant Table. The set increment from the Constant Table is applied to the first available slot's complete set count, to give the number of complete sets until the send time. The time within a partial set from the Constant Table can be used directly; it is the partial set send time.

If more than one slot was reserved, the complet calculation is only needed for the first slot. After that, the send times for the others are generated by simply indexing through the Send Time Constant Table. The large slot send time calculation algorithm is:

1. Set N = number of slots reserved by this reservation.
2. Set COUNT = 0.
3. Set QCNT = current queue count.
4. If QCNT<5, set QCNT = 5.
5. Set WORKQ = QCNT + 1 (gives the next available large slot reachable at this time).
6. Calculate the number of complete sets until the next available large slot:

   $$S = [WORKQ \div 5]_{quotient}$$

7. Calculate the slot number within a partial set:

   $$I = [WORKQ]_{remainder}+1$$

8. Get the I^{th} entry from the Send Time Constant Table, namely the set increment (I), and the time within a set(I).
9. Compute S' = S + set increment(I).
10. Store S' in the number of sets column of the Send Time Table, at entry IN.
11. Store the time within a set(I) in the Send Time Table, also at entry IN.
12. Set IN = IN + 1. If IN points past the end of the Send Time Table, set IN = 1.
13. Set I = I + 1.
14. Set COUNT = COUNT + 1.
15. If COUNT = N, go to step 16. Otherwise go to step 8.

16. QCNT = QCNT + N. Stop.

A SIMP only needs to compute and save the send times of its own slots. Its queue count, however, must include everyone's slots. The algorithm for updating the current queue count, QCNT, after someone else makes a reservation is:

1. Set N = number of slots reserved by the reservation.

2. If QCNT<5, set QCNT = 5.

3. Compute the new queue count as

   QCNT = QCNT + N.

4. Stop.

To reflect elapsed time, four things occur when the satellite's synchronization signal interrupts the SIMP:

1) The clock in the SIMP is reset to zero.

2) One is subtracted from two special registers (described below).

3) Five is subtracted from the current queue count to account for packets that were sent during the last cycle. If the result is less than zero, it is set to zero.

4) One is subtracted from every non-zero set count in the Send Time Table.

After these four things have been done, any new reservations arriving in the small slots are added to the Send Time Table and/or the queue count.

The procedure to transmit a large packet can now be outlined. The description assumes that there is a special register whose contents are automatically compared against the clock. When the register value equals the clock value, an interrupt is generated.

The SIMP loads the special register with the Send Time Table entry point to by the OUT pointer. If the set count is not equal to zero, it will not match the clock, which, because it is reset with every synchronization signal (step 1 above), counts time within a set. As each set goes by, the set count in the special register will be decremented by one (step 2 above). When the clock and the special register are equal, the SIMP will process the resulting interrupt by initiating the DMA to send out the first packet on the "ready to be transmitted" list. It will then increment the OUT pointer and reload the special register. The SIMP is then free to return to its previous function until the DMA transfer is complete. At that time the SIMP must move the packet to the "waiting for an acknowledgment" list. When it is time to send out the next packet, another interrupt will occur.

Throughout this discussion we have only indicated how the send times for large slots are calculated. The send time for small slots must be computed, also. For three reasons, it is not convenient to put the small slot send times in the same Send Time Table as those for large slots. First of all, a flag would be needed to tell which kind of packet to send. Secondly, putting them both in the same table complicates the algorithm for generating large slot send times, while still keeping the table in order of increasing time. Thirdly, the send time for small packets does not vary from set to set. To be included in the regular Send Time Table would result in many redundant entries.

This implementation proposes using a second special register, like the first, which is also compared against the clock. (Note that the two special registers will never contain the same value.) For small slot send time determination, the SIMP can use a Mini Send Time Table which always has set counts of zero, and fixed send times. Only the OUT pointer needs to be moved. (There is no IN pointer, since it is a constant table.) See Figure 21.

A procedure similar to that for large slots is used for small packet timing. The SIMP loads the second special register with a value from the

	Complete Sets	Time Within A Set
	0	.0540 ±%
	0	.0594 ±%
OUT →	0	.0702 ±%
	0	.0648 ±%
	0	.0756 ±%

Figure 21. Mini Send Time Table

Mini Table. When the interrupt occurs, the SIMP checks the "small packets to be transmitted" list. If there is nothing on it, the SIMP reloads the special register with the same value, without changing the OUT pointer. That will cause it not to interrupt until the next set. Similarly, if there is only one small packet waiting, the SIMP will initiate the DMA to send it, but will reload the register without changing the pointer. If more than one packet is waiting, however, the SIMP will increment the OUT pointer, and will load a different value into the register after each packet. This causes multiple interrupts during the set. Although there are five small slots, so that five small packets could be sent out, blocking would be very probable if every SIMP were doing the same thing. It seems better to limit each SIMP to two small slots, say, and also to arrange their Mini Tables in a variety of orders, so the choices do not always overlap.

5.6 Overall Control

Many types of processing have been described in the previous sections. Still necessary is a controlling superstructure to insure that all the functions get performed at the proper times.

The major functions discussed so far can be divided into two groups, those which are initiated by an interrupt, and those which are not:

Interrupt-Driven Functions	Source of Interrupt
1. Prepare for small slots	Satellite synchronization signal
2. Begin receiving a small packet	Receiver gets data
3. Begin sending a small packet	Clock matches special register
4. Begin receiving a large packet	Receiver recognizes header destination field
5. Begin sending a large packet	Clock matches special register
6. All data transferred in; process the packet	DMA complete

7. All data transferred out; move the packet to another list — DMA complete

Non-Interrupt Functions

8. Poll hosts and read in data

9. Process a completed string of formatted packets from a host

10. Send data to a host belonging to the SIMP

11. Execute the utility routine which identifies abnormal delays

Miscellaneous Non-Interrupt Function

12. Cold start procedure to initialize header cells, the ASL, etc.

We now introduce the two programs which implement the controlling superstructure, a Master Scheduler and an Express Scheduler. The Master Scheduler (MS) will be in control during long time slots. By referencing a table of program starting addresses, the MS can initiate execution of any of the non-interrupt function programs. When one of these function programs begins execution, the first thing that program does is put its identifier on a function stack. When the routine is finished, it takes its identifier off the stack and returns control to the MS. The MS then decides which program should execute next and starts it. Execution continues in this manner until an interrupt occurs.

When an interrupt occurs, standard interrupt processing will begin automatically. It is convenient to specify that the SIMPs have interrupt hardware which puts PSW's on a stack. Therefore, the current PSW will be stored on the PSW stack, a new PSW, containing the starting address of an interrupt handler routine, will be loaded, and the interrupt handler routine will begin execution. In the case of interrupts 2 through 7 in the preceeding list, the interrupt handler routine will be the program which performs the function listed. Like the non-interrupt routines, these also add and remove their identifiers from the function stack mentioned above.

The Express Scheduler (ES), the second of the programs which form the controlling superstructure, is the interrupt handler routine for the satellite synchronization signal. Like any interrupt handler, it puts its identifier on the function stack when it starts, and will return control to the MS when it is done. The ES retains control while short packets are arriving at a SIMP. Since all five small slots may not have been used, the ES will have to set a timer to know when the end of the short slots is. It cannot just count five packets and then return control to the MS.

The ES has several housekeeping chores. It is the routine which puts the receiver hardware in non-selective mode and which decrements all non-zero set counts in the Send Time Table. An additional task is to add one to a utility count, whose purpose will be described below.

After the housekeeping is done, the ES sits in a wait state until a small packet arrives. This wait could be considered a waste of computer time if nothing arrives right away. However, the small slots are only .0054 seconds long. When a small packet does arrive, as much time as possible should be available to process it before the next packet arrives. If a packet arrives before the housekeeping is done, it will interrupt the ES. The housekeeping will be finished after the packet has been processed.

When a small packet arrives, it interrupts the ES, causing it to fall out of the wait state or temporarily suspend housekeeping. The ES instead begins processing the packet, which includes the following decision process: Is this packet either from or for this SIMP? If no, the

ES just discards it, and resumes its previous processing until the next packet arrives. If yes, the ES checks if the packet was blocked. For blocked packets from this SIMP, the ES gives control to the retransmit routine. For non-blocked packets for this SIMP, the ES checks the packet type code, and transfers to the appropriate routine.

The processing for most small packet types is minimal, and should be completed before the next packet arrives. Reservation packets, however, might take more time. The routine which handles them can save all the pertinent data, and finish the processing later after the last small slot. All the small packet handling routines return control to the ES. After the small-slot interval is ended and all the packets are processed, the ES removes its identifier from the function stack and gives control back to the MS.

The purpose of the function stack, mentioned throughout this discussion, will now be described. In most computer systems, when an interrupt occurs, the old PSW is stored and an interrupt handler routine begins execution. When the interrupt routine is done, the old PSW is reloaded, and the original program continues its processing. However, in this implementation we do not always want automatically to restart the original program. In an attempt to equalize the service to all of a SIMP's hosts, the two routines which read data from a host or send data to a host are not restarted. They always start at the beginning, communicating with the next host in line in the polling list.

To intercept automatic restarts after an interrupt, all interrupt handler routines (including the ES) branch to the MS when they are done, instead of doing a normal return. Then the MS looks at the identifier on the top of the function stack, since it corresponds to the program which was interrupted. If the identifier indicates a routine which should be resumed, the MS reloads the old PSW from the PSW stack, and the program will continue from where it was when interrupted. Otherwise, if the top identifier specifies one of the programs which is not to be restarted, the MS throws away the top entries from both the PSW and the function stacks. The MS then checks the new top entry on the function stack, if any.

If the function stack is empty, the MS will start a non-interrupt function program. Normally the MS will alternately start the "send to host" or "read from host" program. Whenever the utility count (incremented by the ES) reaches a specified number, however, the MS will start the utility program instead. The utility count will be set back to zero.

One non-interrupt function program is not started by the MS. That program decides whether or not a string of completed packets can go on the "to be transmitted" list and creates reservation or request for status packets (function #9 in the list at the beginning of this section). The MS does not have enough information to decide when this program is needed. Only routine #8, which reads data in from a host, knows when an entire message has been formatted, so it calls routine #9. Routine #9 puts its own identifier on the function stack. If it is interrupted, it will resume processing after the interrupt is serviced in the usual manner.

Figure 22 gives an example of sequencing and interrupt handling. At A, the SIMP is reading data in from a host. At B, an interrupt has occurred, indicating that it is time to send a large packet. The transmit routine has begun executing. The transmit routine will leave the interrupt system disabled for the few instructions which actually start the outbound DMA. While the interrupts are disabled for the transmission, the inbound DMA, which moves data from the receiver into memory, may complete a transfer. As soon as the interrupt system is enabled, the DMA

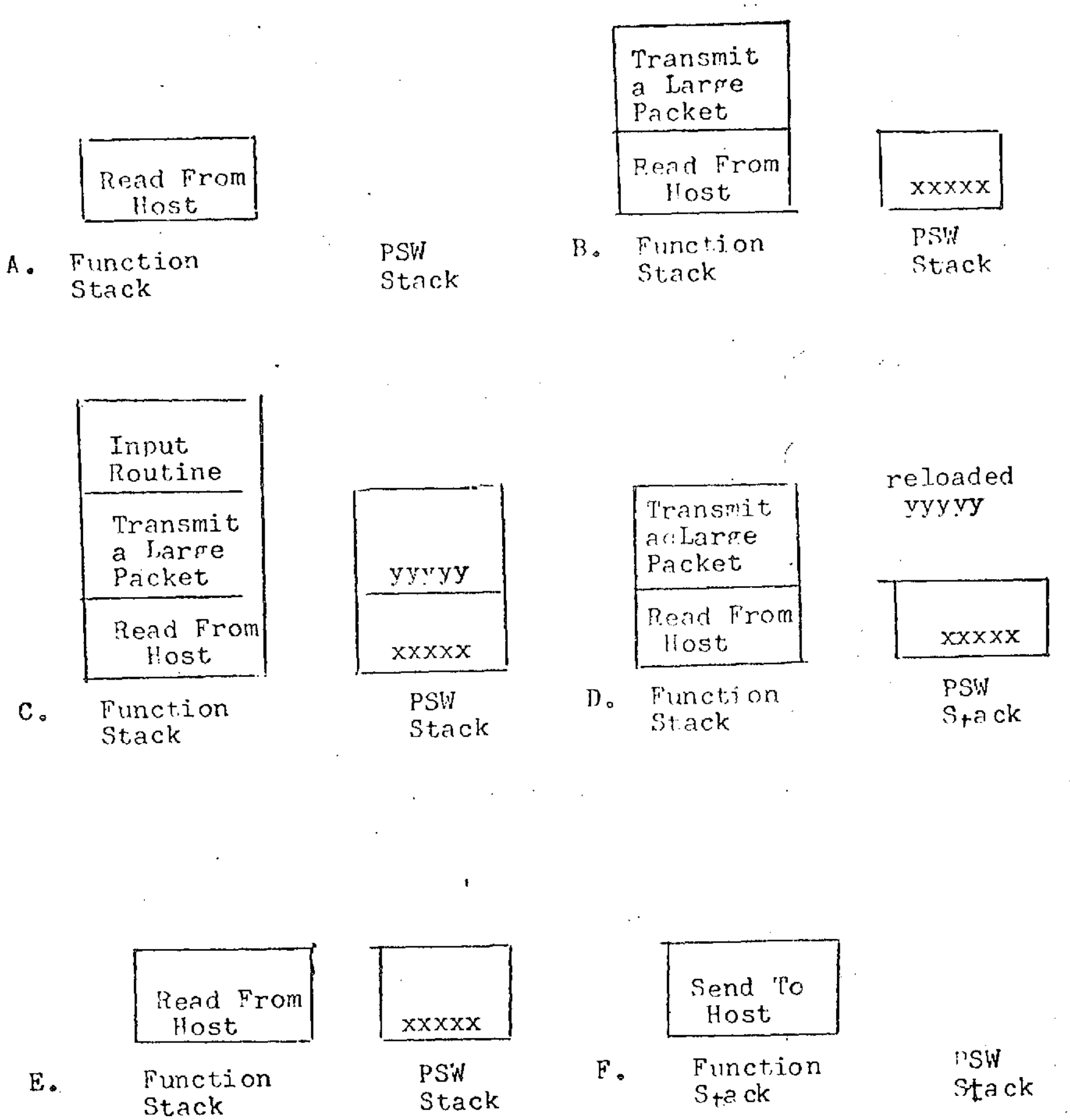


Figure 22. Example of Sequencing and Interrupt Handling

complete interrupt will cause activation of the input routine. The resulting stack contents are shown in C.

When the input routine has completed its processing, it removes its identifier from the stack, and branches to the MS. The MS checks the top entry, and reloads the old PSW, as shown in D of Fig. 22. When the transmit routine is done, however, the "read from host" identifier will be on the top of the function stack (E). The MS will discard it and the corresponding PSW value. Since the stack is now empty, the MS will initiate the "send to host" routine (F).

REFERENCES

1. H. Falk, "Data Communications", IEEE Spectrum, Jan. 1974, Vol. 11, No. 1, pp. 36-39.

2. D. Christiansen, "Technology '74", IEEE Spectrum, Jan. 1974, Vol. 11, No. 1, pp. 30-31.

3. F. Heart, R. Kahn, S. Orstein, W. Crowther, D. Walden, "The Interface Message Processor for the ARPA Computer Network", AFIPS Conference Proceedings, May 1970, Vol. 36, pp. 5510568.

4. N. Abramson, "The Aloha System", Computer-Communication Networks, Prentice-Hall, Inc., 1973, pp. 501-517.

5. J. Martin, Systems Analysis for Data Transmission, Prentice-Hall, Inc., 1972.

6. "Datacomm Developments", The Data Communications User, November 1973, pp. 51.

7. F. Kuo, R. Binder, "Computer-Communications by Radio and Satellite: The Aloha System", Proceedings of the International Advanced Study Institute on Computer Communications Networks, University of Sussex, Sept. 9-15, 1973.

8. N. Abramson, "Packet Switching With Satellites", AFIPS Conference Proceedings, June 1973, Vol. 42, pp. 695-702.

9. The Datacomm Planner, August 1973, pp. 18-19.

10. F. Kuo, N. Abramson, Computer-Communications Networks, Prentice-Hall, Inc., 1973.

11. L. Roberts, "Dynamic Allocation of Satellite Capacity Through Packet Reservation", AFIPS Conference Proceedings, June 1973, Vol. 42, pp. 711-716.

12. H. Inose, "Communication Networks", Scientific American, Sept. 1972, Vol. 227, No. 3, pp. 117-128.

13. L. Roberts, "Aloha Packet System With and Without Slots and Capture", ARPA Satellite System Note 8, June 26, 1972.

14. W. Crowther, R. Rettberg, F. Heart, S. Ornstein, D. Walden, "A System for Broadcast Communication: BBN Aloha", ARPA Satellite System Note 21, Nov. 10, 1972.

15. ITOS--Night-Day Meteorological Satellite, prepared for NASA Goddard Space Flight Center, U.S. Government Printing Office, Washington, D. C.

16. J. Kersey, "Taking a Fresh Look at Data Link Controls", Data Communications Systems, Sept. 1973, pp. 65-71.